[re](https://docs.python.org/3/library/re.html" \l "module-re" \o "re: Regular expression operations.) — Regular expression operations

A regular expression (or RE) specifies a set of strings that matches it; the functions in this module let you check if a particular string matches a given regular expression (or if a given regular expression matches a particular string, which comes down to the same thing).Regular expressions (called REs, or regexes, or regex patterns) are essentially a tiny, highly specialized programming language embedded inside Python and made available through the re module.

### Matching Meta Characters

. ^ $ \* + ? { } [ ] \ | ( )

1. .

(Dot.) In the default mode, this matches any character except a newline. If the [DOTALL](https://docs.python.org/3/library/re.html#re.DOTALL) flag has been specified, this matches any character including a newline.

1. ^

(Caret.) Matches the start of the string, and in [MULTILINE](https://docs.python.org/3/library/re.html#re.MULTILINE) mode also matches immediately after each newline.

1. $

Matches the end of the string or just before the newline at the end of the string, and in [MULTILINE](https://docs.python.org/3/library/re.html#re.MULTILINE) mode also matches before a newline. foo matches both ‘foo’ and ‘foobar’, while the regular expression foo$ matches only ‘foo’. More interestingly, searching for foo.$ in 'foo1\nfoo2\n' matches ‘foo2’ normally, but ‘foo1’ in [MULTILINE](https://docs.python.org/3/library/re.html#re.MULTILINE) mode; searching for a single $ in 'foo\n' will find two (empty) matches: one just before the newline, and one at the end of the string.

1. \*

Causes the resulting RE to match 0 or more repetitions of the preceding RE, as many repetitions as are possible. ab\* will match ‘a’, ‘ab’, or ‘a’ followed by any number of ‘b’s.

1. +

Causes the resulting RE to match 1 or more repetitions of the preceding RE. ab+ will match ‘a’ followed by any non-zero number of ‘b’s; it will not match just ‘a’.

1. ?

Causes the resulting RE to match 0 or 1 repetitions of the preceding RE. ab? will match either ‘a’ or ‘ab’.

1. {m, n}

Causes the resulting RE to match from m to n repetitions of the preceding RE, attempting to match as many repetitions as possible. For example, a{3,5} will match from 3 to 5 'a' characters. Omitting m specifies a lower bound of zero, and omitting n specifies an infinite upper bound. As an example, a{4,}b will match 'aaaab' or a thousand 'a' characters followed by a 'b', but not 'aaab'.

1. \

Either escapes special characters (permitting you to match characters like '\*', '?', and so forth), or signals a special sequence; special sequences are discussed below.

If you’re not using a raw string to express the pattern, remember that Python also uses the backslash as an escape sequence in string literals; if the escape sequence isn’t recognized by Python’s parser, the backslash and subsequent character are included in the resulting string. However, if Python would recognize the resulting sequence, the backslash should be repeated twice. This is complicated and hard to understand, so it’s highly recommended that you use raw strings for all but the simplest expressions.

1. []

Characters can be listed individually, e.g. [amk] will match 'a', 'm', or 'k'.

Ranges of characters can be indicated by giving two characters and separating them by a '-', for example [a-z] will match any lowercase ASCII letter, [0-5][0-9] will match all the two-digits numbers from 00 to 59, and [0-9A-Fa-f] will match any hexadecimal digit. If - is escaped (e.g. [a\-z]) or if it’s placed as the first or last character (e.g. [-a] or [a-]), it will match a literal '-'.

1. |

A|B, where A and B can be arbitrary REs, creates a regular expression that will match either A or B. An arbitrary number of REs can be separated by the '|' in this way.

1. (...)

Matches whatever regular expression is inside the parentheses, and indicates the start and end of a group; the contents of a group can be retrieved after a match has been performed, and can be matched later in the string with the \number special sequence.

**Escape Sequences**

1. \A

Matches only at the start of the string.

1. \b

Matches the empty string, but only at the beginning or end of a word. A word is defined as a sequence of word characters. Note that formally, \b is defined as the boundary between a \w and a \W character (or vice versa), or between \w and the beginning/end of the string. This means that r'\bfoo\b' matches 'foo', 'foo.', '(foo)', 'bar foo baz' but not 'foobar' or 'foo3'.

1. \B

Matches the empty string, but only when it is not at the beginning or end of a word. This means that r'py\B'matches 'python', 'py3', 'py2', but not 'py', 'py.', or 'py!'. \B is just the opposite of \b, so word characters in Unicode patterns are Unicode alphanumerics or the underscore, although this can be changed by using the [ASCII](https://docs.python.org/3/library/re.html#re.ASCII) flag. Word boundaries are determined by the current locale if the [LOCALE](https://docs.python.org/3/library/re.html#re.LOCALE) flag is used.

1. \d

Matches any Unicode decimal digit (that is, any character in Unicode character category [Nd]). This includes [0-9], and also many other digit characters. If the [ASCII](https://docs.python.org/3/library/re.html#re.ASCII) flag is used only [0-9] is matched.

1. \D

Matches any character which is not a decimal digit. This is the opposite of \d. If the [ASCII](https://docs.python.org/3/library/re.html#re.ASCII) flag is used this becomes the equivalent of [^0-9].

1. \s

Matches Unicode whitespace characters (which includes [ \t\n\r\f\v], and also many other characters, for example the non-breaking spaces mandated by typography rules in many languages). If the [ASCII](https://docs.python.org/3/library/re.html#re.ASCII) flag is used, only [ \t\n\r\f\v] is matched.

1. \S

Matches any character which is not a whitespace character. This is the opposite of \s. If the [ASCII](https://docs.python.org/3/library/re.html#re.ASCII) flag is used this becomes the equivalent of [^ \t\n\r\f\v].

1. \w

Matches Unicode word characters; this includes most characters that can be part of a word in any language, as well as numbers and the underscore. If the [ASCII](https://docs.python.org/3/library/re.html#re.ASCII) flag is used, only [a-zA-Z0-9\_] is matched.

1. \W

Matches any character which is not a word character. This is the opposite of \w. If the [ASCII](https://docs.python.org/3/library/re.html#re.ASCII) flag is used this becomes the equivalent of [^a-zA-Z0-9\_]. If the [LOCALE](https://docs.python.org/3/library/re.html#re.LOCALE) flag is used, matches characters considered alphanumeric in the current locale and the underscore.

1. \Z

Matches only at the end of the string.

## Module Contents 1) re.compile(pattern, flags=0)

Compile a regular expression pattern into a [regular expression object](https://docs.python.org/3/library/re.html#re-objects), which can be used for matching using its [match()](https://docs.python.org/3/library/re.html#re.Pattern.match), [search()](https://docs.python.org/3/library/re.html#re.Pattern.search) and other methods, described below.

The expression’s behaviour can be modified by specifying a flags value. Values can be any of the following variables, combined using bitwise OR (the | operator).

The sequence

prog = re.compile(pattern)

result = prog.match(string)

but using [re.compile()](https://docs.python.org/3/library/re.html#re.compile) and saving the resulting regular expression object for reuse is more efficient when the expression will be used several times in a single program.

2) re.**match**(pattern, string, flags=0)

If zero or more characters at the beginning of string match the regular expression pattern, return a corresponding [match object](https://docs.python.org/3/library/re.html#match-objects). Return None if the string does not match the pattern; note that this is different from a zero-length match.

Note that even in [MULTILINE](https://docs.python.org/3/library/re.html#re.MULTILINE) mode, [re.match()](https://docs.python.org/3/library/re.html#re.match) will only match at the beginning of the string and not at the beginning of each line.

3) re.**fullmatch**(pattern, string, flags=0)

If the whole string matches the regular expression pattern, return a corresponding [match object](https://docs.python.org/3/library/re.html#match-objects). Return Noneif the string does not match the pattern; note that this is different from a zero-length match.

4) re.**split**(pattern, string, maxsplit=0, flags=0)

Split string by the occurrences of pattern. If capturing parentheses are used in pattern, then the text of all groups in the pattern are also returned as part of the resulting list. If maxsplit is nonzero, at most maxsplit splits occur, and the remainder of the string is returned as the final element of the list.

**>>>** re.split(r'\W+', 'Words, words, words.')

['Words', 'words', 'words', '']

**>>>** re.split(r'(\W+)', 'Words, words, words.')

['Words', ', ', 'words', ', ', 'words', '.', '']

**>>>** re.split(r'\W+', 'Words, words, words.', 1)

['Words', 'words, words.']

**>>>** re.split('[a-f]+', '0a3B9', flags=re.IGNORECASE)

['0', '3', '9']

If there are capturing groups in the separator and it matches at the start of the string, the result will start with an empty string. The same holds for the end of the string:

**>>>** re.split(r'(\W+)', '...words, words...')

['', '...', 'words', ', ', 'words', '...', '']

That way, separator components are always found at the same relative indices within the result list.

Empty matches for the pattern split the string only when not adjacent to a previous empty match.

**>>>** re.split(r'\b', 'Words, words, words.')

['', 'Words', ', ', 'words', ', ', 'words', '.']

**>>>** re.split(r'\W\*', '...words...')

['', '', 'w', 'o', 'r', 'd', 's', '', '']

**>>>** re.split(r'(\W\*)', '...words...')

['', '...', '', '', 'w', '', 'o', '', 'r', '', 'd', '', 's', '...', '', '', '']

5) re.**findall**(pattern, string, flags=0)

Return all non-overlapping matches of pattern in string, as a list of strings. The string is scanned left-to-right, and matches are returned in the order found. If one or more groups are present in the pattern, return a list of groups; this will be a list of tuples if the pattern has more than one group. Empty matches are included in the result.

6) re.**finditer**(pattern, string, flags=0)

Return an [iterator](https://docs.python.org/3/glossary.html#term-iterator) yielding [match objects](https://docs.python.org/3/library/re.html#match-objects) over all non-overlapping matches for the RE pattern in string. The string is scanned left-to-right, and matches are returned in the order found. Empty matches are included in the result.

7) re.**sub**(pattern, repl, string, count=0, flags=0)

Return the string obtained by replacing the leftmost non-overlapping occurrences of pattern in string by the replacement repl. If the pattern isn’t found, string is returned unchanged. repl can be a string or a function; if it is a string, any backslash escapes in it are processed. That is, \n is converted to a single newline character, \r is converted to a carriage return, and so forth. Unknown escapes such as \& are left alone. Back references, such as \6, are replaced with the substring matched by group 6 in the pattern.

8) re.**subn**(*pattern*, *repl*, *string*, *count=0*, *flags=0*)

Perform the same operation as [sub()](https://docs.python.org/3/library/re.html#re.sub), but return a tuple (new\_string, number\_of\_subs\_made).

9) re.**escape**(*pattern*)

Escape special characters in *pattern*. This is useful if you want to match an arbitrary literal string that may have regular expression metacharacters in it. For example:

**>>>** print(re.escape(r'python.exe'))

python\.exe

10) re.**purge**()

Clear the regular expression cache

**Regex Flags**

1. re.ASCII(re.A)

Make \w, \W, \b, \B, \d, \D, \s and \S perform ASCII-only matching instead of full Unicode matching. This is only meaningful for Unicode patterns, and is ignored for byte patterns. Corresponds to the inline flag (?a).

Note that for backward compatibility, the re.U flag still exists (as well as its synonym re.UNICODE and its embedded counterpart (?u)), but these are redundant in Python 3 since matches are Unicode by default for strings (and Unicode matching isn’t allowed for bytes).

1. re. DOTALL(re.S)

Make the '.' special character match any character at all, including a newline; without this flag, '.' will match anything except a newline. Corresponds to the inline flag (?s).

1. re. IGNORECASE(re.I)

Perform case-insensitive matching; expressions like [A-Z] will also match lowercase letters. Full Unicode matching (such as Ü matching ü) also works unless the [re.ASCII](https://docs.python.org/3/library/re.html#re.ASCII) flag is used to disable non-ASCII matches. The current locale does not change the effect of this flag unless the [re.LOCALE](https://docs.python.org/3/library/re.html#re.LOCALE) flag is also used. Corresponds to the inline flag (?i).

Note that when the Unicode patterns [a-z] or [A-Z] are used in combination with the [IGNORECASE](https://docs.python.org/3/library/re.html#re.IGNORECASE) flag, they will match the 52 ASCII letters and 4 additional non-ASCII letters: ‘İ’ (U+0130, Latin capital letter I with dot above), ‘ı’ (U+0131, Latin small letter dotless i), ‘ſ’ (U+017F, Latin small letter long s) and ‘K’ (U+212A, Kelvin sign). If the [ASCII](https://docs.python.org/3/library/re.html#re.ASCII) flag is used, only letters ‘a’ to ‘z’ and ‘A’ to ‘Z’ are matched.

1. re. LOCALE(re.L)

Make \w, \W, \b, \B and case-insensitive matching dependent on the current locale. This flag can be used only with bytes patterns. The use of this flag is discouraged as the locale mechanism is very unreliable, it only handles one “culture” at a time, and it only works with 8-bit locales. Unicode matching is already enabled by default in Python 3 for Unicode (str) patterns, and it is able to handle different locales/languages. Corresponds to the inline flag (?L).

1. re. MULTILINE(re.M)

When specified, the pattern character '^' matches at the beginning of the string and at the beginning of each line (immediately following each newline); and the pattern character '$' matches at the end of the string and at the end of each line (immediately preceding each newline). By default, '^' matches only at the beginning of the string, and '$' only at the end of the string and immediately before the newline (if any) at the end of the string. Corresponds to the inline flag (?m).

1. re. VERBOSE(re.X)

This flag allows you to write regular expressions that look nicer and are more readable by allowing you to visually separate logical sections of the pattern and add comments. Whitespace within the pattern is ignored, except when in a character class, or when preceded by an unescaped backslash, or within tokens like \*?, (?: or (?P<...>. When a line contains a # that is not in a character class and is not preceded by an unescaped backslash, all characters from the leftmost such # through the end of the line are ignored.